Learning union of k-testable languages

Statistical and symbolic language modeling project

Rania el Bouhssini, Martin Laville and Félix Jamet

December 22, 2018

Contents

1 Introduction

Unless explicitely specified, all definitions and algorithms in this document are coming from Linard et al. [1].

We will present a possible implementation of those definitions and algorithms in a modular fashion, using Python3. Modular meaning here that we will implement concepts as they come and assemble them later as a whole when the necessary parts are complete. So if an `__init__` appears in the wild without its enclosing `class`, it's nothing to worry about.

2 k-testable languages

A k-testable language is a language that can be recognised by sliding a window of size k over an input. By definition, we have $k > 0$ since sliding a window of size zero or less would hardly make any sense.

All necessary informations to recognise such a language can be stored in a k-test vector.

2.1 k-test vector

A k-test vector is a 4-tuple $Z = \langle I, F, T, C \rangle$:

— $I \in \Sigma^{k-1}$ is a set of allowed prefixes,
— $F \in \Sigma^{k-1}$ is a set of allowed suffixes,
— $T \in \Sigma^{k}$ is a set of allowed segments, and
— $C \in \Sigma^{<k}$ is a set of allowed short strings satisfying $I \cap F = C \cap \Sigma^{k-1}$.

We will refer to I, F, T and C respectively as the allowed prefixes, suffixes, infixes and short strings. An intuitive way to formulate the constraint on short strings is that the short strings of length $k-1$ have to be both prefixes and suffixes and vice versa.

This definition can be translated into an init.

`Init k-test vector`:

```
def __init__(self, prefixes, suffixes, infixes, shorts):
    self.k = len(next(iter(prefixes))) + 1
    self.prefixes = prefixes
    self.suffixes = suffixes
    self.infixes = infixes
    self.shorts = shorts
    self.ensure_correct_definition()
```

We then write `ensure_correct_definition` to make sure that the created k-test vector respects the conditions of the definition.

Ensure correct definition:

```
def ensure_correct_definition(self):
    def same_length(collection, reference_length):
        return all(map(lambda x: len(x) == reference_length, collection))

    errors = []
    if not same_length(self.prefixes, self.k - 1):
        errors.append('incorrect prefix length')
    if not same_length(self.suffixes, self.k - 1):
        errors.append('incorrect suffix length')
    if not same_length(self.infixes, self.k):
        errors.append('incorrect infix length')
    if not all(map(lambda x: len(x) < self.k, self.shorts)):
        errors.append('incorrect short string length')
    presufixes = self.prefixes & self.suffixes
    shorts_len_k = set(filter(lambda x: len(x) == self.k - 1, self.shorts))
    if presufixes != shorts_len_k:
        errors.append('short strings conditions not satisfied')

    if len(errors) >0:
        raise ValueError(', '.join(errors).capitalize() + '.')
```

2.2 k-test vectors as a partially ordered set

Let $\mathcal{T}_k$ be the set of all k-test vectors. A partial order $\sqsubseteq$ can be defined on $\mathcal{T}_k$ as follow:

$$\langle I, F, T, C\rangle \sqsubseteq \langle I', F', T', C'\rangle \iff I \subseteq I' \wedge F \subseteq F' \wedge T \subseteq T' \wedge C \subseteq C'$$

With this partial order, a union, an intersection and a symmetric difference can be defined on the k-test vectors $Z = \langle I, F, T, C\rangle$ and $Z' = \langle I', F', T', C'\rangle$.

2.2.1 Union $\sqcup$

$$Z \sqcup Z' = \langle I \cup I', F \cup F', T \cup T', C \cup C' \cup (I \cap F') \cup (I' \cap F)\rangle$$

We can see that the constraint on short strings $I \cap F = C \cap \Sigma^{k-1}$ is still respected because the short strings are updated with all the cases which could contradict it.

The implementation is a quite litteral translation.

K-test vector union:

```
def union(self, other):
    prefixes = self.prefixes | other.prefixes
    suffixes = self.suffixes | other.suffixes
    infixes = self.infixes | other.infixes
    shorts = self.shorts | other.shorts |\
        (self.prefixes & other.suffixes) |\
        (self.suffixes & other.prefixes)
    return ktestable(prefixes, suffixes, infixes, shorts)
```

2.3 Putting the pieces together

All the blocks seen previously are simply put together in the `ktestable` class.

```
class ktestable(object):
    <<Init k-test vector>>

    <<Ensure correct definition>>

    <<K-test vector union>>
```

2.4 Tests

We put together some tests to ensure that the implementation works at least superficially as intended:

```
tests = {
    'invalid example': ({'aa'}, {'aa'}, {'aaaa'}, {'ada'}),
    'aa+': ({'aa'}, {'aa'}, {'aaa'}, {'aa'}),
    'bb+': ({'bb'}, {'bb'}, {'bbb'}, {'bb'})
}
instanciations = {}

for name, parameters in tests.items():
    try:
        ktest = ktestable(*parameters)
        print('The creation of %s went well' % name)
        instanciations[name] = ktest
    except ValueError as e:
        print('The creation of %s failed:\n - ' % name, e)
    print()

union = instanciations['aa+'].union(instanciations['bb+'])
print(union.prefixes)
```

The creation of invalid example failed:

— Incorrect infix length, incorrect short string length, short strings conditions not satisfied.

The creation of aa+ went well

The creation of bb+ went well

{'bb', 'aa'}

3 Sources

1. Linard, A., de la Higuera C., Vaandrager F.:Learning Unions of k-Testable Languages (`http://www.sws.cs.ru.nl/publications/papers/fvaan/kTestable/main.pdf`)